平成 24 年度 卒業論文概要			
所　　属	機械情報工学科	指導教員	光來 健一
学生番号	10237209	学生氏名	西村 謙一郎
論文題目	アスペクト指向プログラミングを用いた IDS のオフロード支援		

1　はじめに

近年、インターネットに接続されたコンピュータに対する不正アクセスが増加している。攻撃者からの不正アクセスを検知するために、不正なプログラムが実行されていないかなどを監視する侵入検知システム（IDS）が用いられるようになってきた。IDS が攻撃されて停止させられる事態を防ぐために、仮想マシン（VM）を用いた IDS のオフロードという手法が提案されている。この手法は IDS と監視対象システムを別々の VM で動かすことで、IDS への攻撃を防ぐことを可能にする。しかし、このようにしてオフロードした IDS（オフロード IDS）を開発するのは労力が大きかった。オフロード IDS が監視対象 VM のカーネルメモリ内部にアクセスするためには、煩雑なプログラミングが必要となるためである。

本研究では、アスペクト指向プログラミング（AOP）を用いたオフロード IDS の開発支援を提案する。

2　オフロード IDS の開発

VM を用いた IDS のオフロードは、監視対象システムを VM 上で動作させ、IDS だけを別の VM（IDS-VM）上で動作させる手法である。IDS をオフロードすることで、監視対象 VM が攻撃を受けたとしても、IDS は正常に監視を行うことが可能となる。IDS-VM は監視のみを行うため、攻撃を受ける可能性を低くすることができる。オフロードした IDS（オフロード IDS）は監視対象 VM 内のシステムを外側から監視する。例えば、実行されているプロセスの一覧をチェックする際には、監視対象 VM の OS カーネルが使っているメモリを調べてプロセスの情報を取得する。

しかし、オフロード IDS を開発するのは容易ではなかった。第一に、オフロード IDS がアクセスするデータが監視対象 VM のメモリ上にあるのか、IDS-VM のメモリ上にあるのかを、開発者が一つ一つ区別する必要がある。例えば、IDS が監視するプロセスの情報は監視対象 VM 上にあるが、IDS が表示するメッセージの文字列は IDS-VM 上にある。第二に、IDS が監視対象 VM のデータにアクセスする箇所すべてについて、専用の関数を用いるようにプログラミングしなければならない。監視対象 VM のメモリにアクセスするには、データのアドレス変換を行ってからメモリをマップするという処理が必要になるためである。特に、大規模な IDS では、このようなプログラミングを正確に行う労力が非常に大きくなる。

3　AOP を用いたオフロード IDS の開発支援

本研究ではアスペクト指向プログラミング（AOP）を用いることにより、オフロード IDS のプログラミングの煩雑さを減らし、開発を容易にする。AOP を用いることで、監視対象 VM のメモリへのアクセスが必要な箇所を自動的に判別し、必要に応じてメモリアクセスのためのコードを実行することができる。これにより、開発者は監視対象 VM の OS カーネル内で動く IDS と同じようにオフロード IDS のプログラミングを行うことができる。

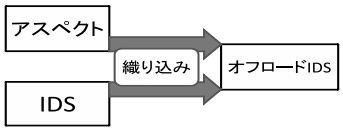

図 1　アスペクトの織り込み

3.1　AOP とは

AOP はプログラムの横断的関心事をモジュール化することを可能にする。横断的関心事とはオブジェクト指向プログラミングではモジュール化できない、他の関心事に依存している関心事である。例えば、オフロード IDS においては監視アルゴリズムが中心となる関心事であり、監視対象 VM のメモリへのアクセスは横断的関心事になる。C 言語等では横断的関心事を一つのモジュールにまとめられず、一部のコードが他の関心事を実装するモジュールの中に散在してしまう。このようなプログラムではコードの変更や修正を加えた場合にミスが発生しやすく、プログラム開発の効率が悪くなってしまう。

AOP では横断的関心事をアスペクトと呼ばれるモジュールに記述し、図 1 のように、ベースとなるプログラムに織り込む。アスペクトはプログラム上の様々な実行点を抽出するためのポイントカットと、抽出された実行点で実行されるコードを定義するためのアドバイスで構成される。例えば、ログを取るためのアスペクトでは、プログラムのどこでログを出力するかをポイントカットで指定し、どのようなログを出力するかをアドバイスで定義する。

3.2　Introspect アスペクト

IDS が必要に応じて監視対象 VM のメモリにアクセスできるようにするための Introspect アスペクトを図 2 に示す。このアスペクトでは、構造体単位で監視対象 VM のメモリへのアクセスかどうかを判別する。そのために、kernel_data というポイントカットを定義して、カーネル構造体のメンバアクセスを抽出している。このポイントカットはデータ参照を実行点として抽出する get ポイントカットを使って定義されている。オフロード IDS は IDS-VM 内で定義される構造体も使うため、監視対象 VM のカーネル構造体をこのポイントカットで指定することによって区別する。

このポイントカットで抽出された実行点では、監視対象

```
aspect Introspect {
  pointcut kernel_data() =
              get("% task_struct::%") ||
              get("% list_head::%");

  advice kernel_data() : around() {
    ulong kaddr = tjp->source();
    ulong base, offset = kaddr & ~PAGE_MASK;
    ulong mfn;

    mfn = xc_translate_foreign_address(kaddr);
    base = xc_map_foreign_range(mfn);

    *tjp->result() = *(base + offset);

    munmap(base, PAGE_SIZE);
  }
};
```

図 2　Introspect アスペクト (抜粋)

```
struct task_struct *init_task =
                      0xffffffff8160b020;
struct task_struct *p = init_task;

do {
  printf("%d %s\n", p->pid, p->comm);
  p = list_entry(p->tasks.next,
                 struct task_struct,
                 tasks);
} while (p != init_task);
```

図 3　プロセス一覧を取得するプログラム（抜粋）

VM のメモリアクセスを行うコードを実行する。そのために、around アドバイスでカーネル構造体のメンバを参照する際に代わりに実行されるコードを定義している。このアドバイスでは、`tjp->source()` が返すカーネル構造体のメンバの仮想アドレスを物理アドレスに変換し、オフロード IDS 内にメモリマップする。次に、対象の構造体メンバの値を読み込み、`tjp->result()` が返すメモリ領域に格納する。

Introspect アスペクトを用いることにより、監視対象 VM のプロセス一覧をカーネルから取得するコードは図 3 のように記述することができる。プロセスリストを init_task からたどっていき、各プロセスのプロセス ID と名前を出力している。このコードはカーネル内でプロセス一覧を取得するコードと init_task の定義を除いて同じである。

3.3　アスペクト処理系

オフロード IDS を C 言語で開発できるように、アスペクト指向言語として AspectC++ [1] を用いた。ただし、C 言語ではポインタやキャストのせいで構造体のメンバアクセスを実行点として完全に抽出するのが難しいため、AspectC++ では get ポイントカットがサポートされていなかった。そこで、

表 1　実験結果

	サイズ増加 [byte]	実行時間 [ms]
関数	106	17.38
マクロ	89	17.55
オーバーロード	225	17.71
AOP	22	21.07

get ポイントカットを限定的にサポートするためのパッチ [2] を適用した。このパッチでは構造体の中の配列のアクセスをうまく扱うことができなかったため、配列アクセスも実行点として正しく抽出できるように修正を加えた。これにより、通常の構造体のメンバアクセスであれば実行点として抽出することができるようになった。

また、AspectC++ はマクロに対応していなかったため、コンパイルの手順を工夫することでマクロを扱えるようにした。まず、C プリプロセッサを用いてオフロード IDS の中で使われているマクロを展開する。アスペクトを織り込む際のマクロの衝突を防ぐために、アスペクトの中で使われているマクロも抽出しておき、同時にオフロード IDS に適用する。そして、マクロを展開したオフロード IDS のプログラムにアスペクトを織り込み、コンパイルを行う。

4　実験

AOP を用いて開発したオフロード IDS について、オフロード前と比べたプログラムサイズの増加量、および、実行時間を測定した。実験には Intel core i7 870、4GB のメモリを搭載したマシンを使用した。仮想化ソフトウェアとして Xen 4.1.1、OS には Linux 2.6.39.3 を用いた。オフロード IDS には図 3 のようなプロセスリストを取得するプログラムを用いた。比較として、関数、マクロ、C++ の演算子のオーバーロード機能を用いて開発したオフロード IDS についても測定を行った。

実験結果は表 1 のようになった。実験の結果より、AOP を用いた場合、他の手法に比べてプログラムサイズが小さくなることが分かった。しかし、実行時間は 3〜4 μ s 程度遅くなった。これは、アスペクトを適用した場合、関数呼び出しが増加することなどが原因だと考えられる。

5　まとめ

本研究では、AOP を用いたオフロード IDS の開発支援を提案した。監視対象 VM のメモリにアクセスする箇所の指定と、そこでメモリアクセスを行うコードをアスペクトとして分離することで、開発者はオフロードのことを意識せずにオフロード IDS の開発を行うことができる。実際に AOP を用いてプロセス情報を取得するオフロード IDS を開発し、提案手法の有用性を示した。今後の課題は、様々なオフロード IDS に適用していくことである。

参考文献

[1] O. Spincxyk et al. AspectC++, `http://www.aspectc.org/`

[2] J.Magnusson. Set and Get in AspectC++, Master's Thesis, 2006.